HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

11-22-00

PATENT APPLICATION

ATTORNEY DOCKET NO. 10001151

IN THE U.S. PATENT AND TRADEMARK OFFICE
Patent Application Transmittal Letter

COMMISSIONER FOR PATENTS
Washington, D.C. 20231

Sir:

Transmitted herewith for filing under 37 CFR 1.53(b) is a(n): (X) Utility ( ) Design
( ) original patent application,
( ) continuation-in-part application

INVENTOR(S): **Carol L. Thompson**

TITLE: **Method and Apparatus for Performing Correctness Checks Opportunistically**

Enclosed are:

(X) The Declaration and Power of Attorney. (X) signed ( ) unsigned or partially signed

(X) 5 sheets of drawings (one set) ( ) Associate Power of Attorney

( ) Form PTO-1449 ( ) Information Disclosure Statement and Form PTO-1449

( ) Priority document(s) (X) (Other) **Assignment** (fee $ 40.00)

CLAIMS AS FILED BY OTHER THAN A SMALL ENTITY				
(1) FOR	(2) NUMBER FILED	(3) NUMBER EXTRA	(4) RATE	(5) TOTALS
TOTAL CLAIMS	10 — 20	0	X $18	$ 0
INDEPENDENT CLAIMS	4 — 3	1	X $80	$ 80
ANY MULTIPLE DEPENDENT CLAIMS	0		$270	$ 0
BASIC FEE: Design ($320.00); Utility ($710.00)				$ 710
TOTAL FILING FEE				$ 790
OTHER FEES				$ 40
TOTAL CHARGES TO DEPOSIT ACCOUNT				$ 830

Charge $ 830 to Deposit Account 08-2025. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16, 1.17,1.19, 1.20 and 1.21. A duplicate copy of this sheet is enclosed.

"Express Mail" label no. EL492180165US

Date of Deposit 11/21/00

I hereby certify that this is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Commissioner for Patents, Washington, D.C. 20231.

By Allison M. Olson

Typed Name: **Allison M. Olson**

Respectfully submitted,

Carol L. Thompson

By Daniel J. Santos

Daniel J. Santos

Attorney/Agent for Applicant(s)
Reg. No. 40,158
Date: 11/21/00

Telephone No.: (770) 933-9500

Rev 10/00 (TransNew)

- Attach as First Page to Transmitted Papers -

METHOD AND APPARATUS FOR PERFORMING CORRECTNESS CHECKS OPPORTUNISTICALLY

TECHNICAL FIELD OF THE INVENTION

The present invention relates to a method and an apparatus for performing correctness checks opportunistically, wherein spare instruction slots within a code module are utilized for the code associated with the correctness checks.

BACKGROUND OF THE INVENTION

When software is developed, the software developer often includes calls to assert functions at certain locations in the source code. When the software program is compiled and executed, the assert functions will be evaluated by the compiler. An assert function is a Boolean statement used in a computer program to test a condition that, if the program is operating correctly, should always evaluate to a certain logic level *(e.g.*, should always evaluate as true or should always evaluate as false). Therefore, if the tested condition evaluates to another logic level, then the assertion test fails, and it is known that an error in the execution of the program has occurred. When an assertion test fails, the program is typically terminated, and an appropriate error message is generated.

Generally, when developing the source code, the developer specifies that the assertions will either all be on or all be off. If the developer specifies that the assertions are all to be off, the compiler will ignore all of the assert functions when generating the instruction schedule. Consequently, at run time, none of the code associated with the assert functions will be executed. Therefore, none of the assert functions will be evaluated. On the other hand, if the developer specifies that all of

the assertions are to be on, the compiler will insert all of the code associated with all of the assert functions into the instruction schedule, regardless of whether or not inserting the assert function code sequences will lengthen the final instruction schedule and thus result in a performance penalty at run time. This is true even in cases where spare instruction slots exist in the initial instruction schedule that would enable at least some of the assert function instructions to be inserted into the schedule without lengthening the schedule.

A function that is typically referred to as a correctness check function is similar to the assert function. Correctness checks are often included in the source code by the developer at locations in the source code where it is desirable to ensure that a value, a range of values, or a relationship between values is correct at a particular point in the code. At run time, the code associated with the correctness check is evaluated. If the result of the evaluation resolves to a non-zero value, then the value, range of values, or relationship between values being evaluated is deemed to be correct.

As with assert functions, simply inserting the code associated with correctness checks into the instruction schedule will result in lengthening the instruction schedule in cases where the number of spare slots existing in the instruction schedule is less than the number of slots needed to accommodate the instructions associated with the correctness checks. Consequently, a performance cost will be realized when the program is executed at run time.

It would be desirable to provide a technique that would enable spare instruction slots existing in the initial instruction schedule to be utilized opportunistically for instructions associated with functions such as assert functions and correctness check functions in such a way that a performance penalty would not

be incurred at run time. In other words, it would be advantageous to provide a way in which the instructions associated with such functions could be inserted into the instruction schedule to the extent that inserting the instructions does not lengthen the instruction schedule. In this way, at least some of the instructions associated with such functions could be executed at run time without necessarily causing a performance cost to be incurred. Accordingly, a need exists for a method and an apparatus that enable these types of functions to be performed by using spare instruction slots within a code module opportunistically.

SUMMARY OF THE INVENTION

The present invention provides a method and an apparatus that enable spare instruction slots within a code module to be utilized opportunistically for insertion of instructions associated with correctness check functions. The apparatus of the present invention comprises a compiler, which may be comprised solely as hardware or as a combination of hardware and software. The compiler performs code generation and generates an initial instruction schedule. During the generation of the initial instruction schedule, the compiler ignores code sequences associated with correctness check functions. After the initial instruction schedule has been generated, the compiler examines the initial instruction schedule and determines locations of spare instruction slots in the initial instruction schedule that can potentially be utilized for insertion of the code sequences associated with the correctness checks. The code sequences associated with the correctness checks are then inserted into the instruction schedule to the extent that insertion of the code sequences does not lengthen the final instruction schedule. Consequently, no performance penalty is incurred at run time.

In accordance with the preferred embodiment of the present invention, the developer writes the source code check sequences so that they are treated as conditions by the compiler. The developer also specifies that the correctness checks are to be performed opportunistically. Once the compiler has performed initial code generation, the compiler will generate an initial instruction schedule utilizing the main instruction stream. The compiler will treat the code associated with the correctness checks specially since the correctness checks are written in the source code as conditions. Therefore, the instructions associated with the correctness checks will not be utilized by the compiler when generating the initial instruction schedule. In essence, by specifying that the correctness checks are conditions when developing the source code, the compiler is informed that the main instruction stream is to be kept separate from the instruction stream associated with the correctness checks when generating the instruction schedule. Therefore, the compiler will only utilize the main instruction stream when generating the initial instruction schedule.

Once the initial instruction schedule has been generated, the compiler analyzes the initial instruction schedule and determines where spare instruction slots exist in the initial instruction schedule. The compiler then determines which correctness check code sequences can be inserted into certain spare instruction slots. The correctness check code sequences are then inserted into the appropriate spare instruction slots to the extent that they can be inserted without lengthening the instruction schedule. Thus, at run time, the correctness checks corresponding to the code sequences that were inserted into the spare instruction slots of the instruction schedule will be executed along with the main code sequences without a performance penalty resulting.

These and other features and advantages of the present invention will become apparent from the following description, drawings and claims.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates the result of performance of initial code generation by the compiler of the present invention during which conditional instructions are kept separate from the main instruction stream.

FIG. 2 illustrates the result of performance of initial instruction scheduling by the compiler of the present invention and shows spare instruction slots in the initial instruction schedule that may or may not be utilized for insertion of correctness check code sequences in accordance with the method of the present invention.

FIG. 3 illustrates the result of performance of final instruction scheduling by the compiler of the present invention during which certain correctness check code sequences are inserted into certain spare instruction slots contained in the initial instruction schedule.

FIG. 4 illustrates the method of the present invention in accordance with the preferred embodiment for opportunistically utilizing spare instruction slots contained in the initial instruction schedule for correctness check code sequences.

FIG. 5 illustrates the apparatus of the present invention, which comprises a central processing unit (CPU) 50 that is capable of being configured to execute the compiler program of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

In accordance with the preferred embodiment of the present invention, when correctness check functions are expressed in the source code by the developer, the

correctness check functions are expressed as conditions. When compilers perform initial code generation, the instructions associated with conditions are kept separate from the main instruction stream. By expressing the correctness checks as conditions in the source code, it is ensured that the instructions associated with the correctness checks will not be contained in the main instruction stream once initial code generation has been performed. This is illustrated in FIG. 1, which shows a stream of conditional instructions that is separate from the main instruction stream 2.

Although the present invention is described herein in relation to utilizing spare instruction slots for code sequences associated with correctness check functions, it should be noted that the present invention is not limited to this particular implementation. As stated above, assert functions and correctness check functions have similarities in terms of the manner in which they are evaluated. Therefore, the present invention could also opportunistically use spare instruction slots for code sequences associated with assert functions. Furthermore, those skilled in the art will understand the manner in which the techniques of the present invention can be utilized with other types of functions for which it would be advantageous to opportunistically utilize spare instruction slots in the instruction schedule in such a way that performance penalties are not incurred at run time. Therefore, the following discussion of the present invention as it relates to correctness check functions should be viewed as an exemplary and preferred embodiment of the present invention and not as an exclusive implementation of the present invention.

Once the task of initial code generation has been performed by the compiler, the task of initial instruction scheduling is performed. FIG. 2 illustrates the results of performance of initial instruction scheduling. The method and apparatus of the present invention preferably are utilized in connection with a processor that issues

multiple instructions per cycle. As shown in FIG. 2, the number of instructions issued per cycle is N+1, where N is some positive integer. For the purposes of describing the present invention, it will be assumed that N is equal to 5. Therefore, the number of instructions issued per cycle will be assumed to be 6, as indicated by the 6 blocks in each of rows 12-25 of the initial instruction schedule shown in FIG. 2.

The shaded blocks in rows 12-25 correspond to instruction slots that are being utilized, i.e., instruction slots that are not spare. The white blocks in rows 12-25 of the instruction schedule correspond to spare instruction slots. The check code sequences 31-35 shown in FIG. 2 correspond to code sequences associated with correctness check functions. It can be seen from FIG. 2 that the initial instruction schedule 10 is separate from the correctness check code sequences 31-35. This results from the separation of the conditional instructions and the main instruction stream when initial code generation is performed.

Once the initial instruction schedule has been generated, the compiler determines where spare instruction slots exist in the initial instruction schedule. The manner in which compilers determine the existence of spare instruction slots in an instruction schedule is generally known in the art. Each correctness check code sequence is associated with a particular sequence of instructions which are contained in the initial construction schedule 10. The compiler knows the relationship between the correctness check code sequences and the main stream code sequences of the initial instruction schedule 10. Therefore, the compiler is capable of determining which correctness check code sequences can be inserted into particular spare instruction slots of the initial instruction schedule 10.

Each instruction slot is not fully general. For example, certain instruction slots may accommodate only memory instructions whereas others may accommodate only

floating point instructions or integer instructions. The compiler understands which slots are capable of accommodating certain types of instructions. For each correctness check code sequence 31-35, the compiler determines whether enough spare instruction slots exist to accommodate the correctness check code sequence. If, for any particular correctness check code sequence, not enough spare instruction slots exist, the correctness check code sequence will be discarded. On the other hand, if enough suitable spare instruction slots exist to accommodate a particular correctness check code sequence, the particular correctness check code sequence will be inserted into the spare instruction slots.

For simplicity and ease of illustration, it will be assumed that each correctness check code sequence shown in FIG. 2 can only be inserted into the instruction slots located in the row across from it in the instruction schedule 10 shown in FIG. 2. Therefore, correctness check code sequence 31 may only be inserted into row 12 of the instruction schedule 10 and only if enough spare instruction slots exist in row 12 to accommodate sequence 31. However, in reality, the instructions of a particular correctness check code sequence could be inserted into different rows of the initial instruction schedule 10 provided all dependencies are satisfied. For example, if the correctness check code sequence corresponds to a comparison of variables x and y, the instructions of the correctness check code sequence must be inserted in such a way that the values of both of the variables x and y are available at the time that the comparison is to be performed in the instruction schedule. Those skilled in the art will understand that the compiler can determine which instructions of a particular correctness check code sequence can be inserted into particular spare instruction slots of the initial instruction schedule.

FIG. 3 illustrates the insertion of correctness check code sequences into the instruction schedule during the generation of the final instruction schedule. The correctness check code sequences that are comprised of more instructions than can be accommodated by spare instruction slots are discarded. For example, correctness check code sequence 31 is comprised of three instructions, but row 12 of the instruction schedule has only one spare instruction slot. Therefore, correctness check code sequence 31 is discarded. In contrast, correctness check code sequence 32 is comprised of only two instructions. Since row 15 contains five spare instruction slots, the entire correctness check code sequence 32 can be accommodated. Therefore, the correctness check code sequence 32 is inserted into the instruction schedule. Similarly, correctness check code sequence 34 can be accommodated by the spare instruction slots contained in row 21 of the instruction schedule. On the other hand, correctness check code sequences 33 and 35 cannot be accommodated and, therefore, are discarded.

Again, the representation illustrated in FIG. 3 is simplistic and is only intended to illustrate that the entire correctness check code sequences must be capable of being accommodated by suitable spare instruction slots or it will be discarded. This ensures that insertion of correctness check code sequences into the instruction schedule will not cause the instruction schedule to be lengthened. Consequently, insertion of the correctness check code sequences into the instruction schedule will not result in a performance cost being incurred at run time.

FIG. 4 illustrates the method of the present invention performed by the compiler of the present invention in accordance with the preferred embodiment. Compilers generally are implemented in the form of computer programs that are executed on some type of hardware platform that is running some type of operating

system. Therefore, in accordance with the preferred embodiment of the present invention, the compiler of the present invention comprises hardware configured to execute a compiler program, as shown in FIG. 5. FIG. 5 simply illustrates a central processing unit (CPU) 50, which is capable of being configured to execute the compiler program of the present invention, and which is in communication with a memory element 51 that stores instructions to be executed by the CPU 50.

However, those skilled in the art will understand that functions that are performed in software typically can also be performed solely in hardware. Therefore, the present invention is not limited to being performed in any specific hardware and/or software. Those skilled in the art will understand that the functions of the present invention can be implemented in a variety of forms.

With reference again to the flow chart shown in FIG. 4, the first step in the method of the present invention performed by the compiler is the code generation task, as indicated by block 41. As stated above, preferably the correctness check functions are included in the source code as conditions so that they will be treated specially by the compiler and kept separate from the main instruction stream, as shown in FIG. 1. It is not required that the correctness check functions be implemented in the source code in the form of conditions. This is merely one way of keeping the correctness check code sequences separate from the main instruction stream until the compiler has determined where correctness check code sequences can be inserted into the instruction schedule. Those skilled in the art will understand that there are other methods that can be used for maintaining the correctness check code sequences separate from the main instruction stream until the compiler has determined where appropriate spare instruction slots exist for insertion of the correctness check code sequences. Furthermore, it is not absolutely necessary that the

correctness check code sequences be kept separate from the main instruction stream, although doing so simplifies the processes that need to be performed by the compiler.

Once the task of code generation has been performed, the initial instruction schedule is generated and the spare instruction slots are located, as discussed above with reference to FIGS. 2 and 3. This step is represented by block 42 in FIG. 4. Once the initial instruction schedule has been generated, the compiler begins selecting correctness check code sequences and determining whether or not enough spare instruction slots exist to accommodate the correctness check code sequences. These steps are represented by blocks 43-47. The compiler selects a check, as indicated by block 43, and determines whether or not enough spare instruction slots exist to accommodate the particular correctness check code sequence, as indicated by block 44. If not, the correctness check code sequence is discarded, as indicated by block 45. If enough spare instruction slots exist in the instruction schedule to accommodate the correctness check code sequence, the correctness check code sequence is inserted into the appropriate spare instruction slots in the instruction schedule, as indicated by block 46. This process continues until the compiler has determined whether each correctness check code sequence can be accommodated by spare instruction slots in the instruction schedule, as indicated by block 47 and the return from that block to block 43.

Of course, if the compiler determines at the step represented by block 42 that no spare instruction slots exist, the tasks represented by blocks 43-47 need not be performed by the compiler. Similarly, if the compiler determines that no more spare instruction slots exist in the instruction schedule after a correctness check code sequence has been inserted into the instruction schedule, the compiler need not continue checking to determine whether subsequent correctness check code sequences

can be inserted into the instruction schedule. Also, the steps represented by blocks 43-47 do not necessarily have to be performed separately or in the order shown in FIG. 4. For example, if a compiler determines that only three spare instruction slots exist and that only one correctness check code sequence can be accommodated by the three spare instruction slots, the compiler may simply insert the correctness check code sequence and then resume its normal operations. In other words, each correctness check code sequence will not have to be analyzed to determine whether or not it can be inserted and a determination will not need to be made as to whether or not the last correctness check code sequence has bee encountered and analyzed by the compiler.

Once the compiler has allocated correctness check code sequences to all of the spare instruction slots capable of accommodating the correctness check code sequences, the compiler may perform other operations, such as optimization. The compiler then executes the compiled program and any correctness check code sequences included in the compiled program will be performed without incurring a performance cost. Certain functions that need to be performed by a compiler in order to perform the method of the present invention are already performed by known compilers. For example, locating spare instruction slots and determining whether a particular instruction or sequence of instructions are capable of being accommodated by the spare instruction slots is a function currently performed by some known compilers. Therefore, configuring a compiler to perform the functions of the present invention should not be overly difficult.

It should be noted that the present invention has been described with reference to preferred embodiments, but that the present invention is not limited to these embodiments. Those skilled in the art will understand that the embodiments

discussed above can be modified without deviating from the scope of the present invention. As stated above, various functions performed in accordance with the present invention are capable of being performed in a variety of ways. Therefore, the present invention is not limited to any particular manner of performing these functions or to any particular physical implementation for performing these functions.

CLAIMS

What is claimed is:

1. An apparatus for performing correctness checks opportunistically, the apparatus comprising:

first logic, the first logic receiving a first set of instructions and generating an initial instruction schedule from the first set of instructions, the first set of instructions including one or more instructions associated with a correctness check function;

second logic, the second logic evaluating the initial instruction schedule to determine whether the initial instruction schedule includes spare instruction slots into which said one or more instructions associated with the correctness check function can be inserted; and

third logic, the third logic inserting said one or more instructions associated with the correctness check function into the spare instruction slots if enough spare instruction slots exist in the initial instruction schedule for accommodating said one or more instructions.

2. The apparatus of claim 1, wherein said one or more instructions associated with the correctness check function correspond to a conditional expression, and wherein the first logic performs initial code generation prior to generating the initial instruction schedule, wherein when the first logic performs initial code generation, said one or more instructions associated with the correctness check function are separated from all other instructions of said first set of instructions so that the initial instruction schedule does not include any instructions associated with the correctness check function.

3. The apparatus of claim 2, wherein said first, second and third logic correspond to a processor programmed to execute a compiler program, the compiler program including a first code segment for performing initial code generation and for generating the initial instruction schedule, a second code segment for evaluating the initial instruction schedule to determine whether spare instruction slots exist in the initial instruction schedule, and a third code segment for inserting said one or more instructions associated with the correctness check function into the spare instruction slots if enough spare instruction slots exist to accommodate said one or more instructions.

4. An apparatus for performing correctness checks opportunistically, the apparatus comprising:

first means for receiving a first set of instructions and for generating an initial instruction schedule from the first set of instructions, the first set of instructions including one or more instructions associated with a correctness check function;

second means for evaluating the initial instruction schedule to determine whether the initial instruction schedule includes spare instruction slots into which said one or more instructions associated with the correctness check function can be inserted; and

third means for inserting said one or more instructions associated with the correctness check function into the spare instruction slots if enough spare instruction slots exist in the initial instruction schedule for accommodating said one or more instructions.

5. The apparatus of claim 4, wherein said one or more instructions associated with the correctness check function correspond to a conditional expression, and wherein the first means performs initial code generation prior to generating the initial instruction schedule, wherein when the first logic performs initial code generation, said one or more instructions associated with the correctness check function are separated from all other instructions of said first set of instructions so that the initial instruction schedule does not include any instructions associated with the correctness check function.

6. A method for performing correctness checks opportunistically, the method comprising the steps of:

receiving a first set of instructions and generating an initial instruction schedule from the first set of instructions, the first set of instructions including one or more instructions associated with a correctness check function;

evaluating the initial instruction schedule to determine whether the initial instruction schedule includes spare instruction slots into which said one or more instructions associated with the correctness check function can be inserted; and

inserting said one or more instructions associated with the correctness check function into the spare instruction slots if enough spare instruction slots exist in the initial instruction schedule for accommodating said one or more instructions.

7. The method of claim 6, wherein said one or more instructions associated with the correctness check function correspond to a conditional expression, and wherein the step of generating the initial instruction schedule includes the step of performing initial code generation, wherein when initial code generation is performed, said one or

more instructions associated with the correctness check function are separated from all other instructions of said first set of instructions so that the initial instruction schedule does not include any instructions associated with the correctness check function.

8. The method of claim 7, wherein the method is performed by a processor programmed to execute a compiler program, the compiler program including a first code segment for performing initial code generation and for generating the initial instruction schedule, a second code segment for evaluating the initial instruction schedule to determine whether spare instruction slots exist in the initial instruction schedule, and a third code segment for inserting said one or more instructions associated with the correctness check function into the spare instruction slots if enough spare instruction slots exist to accommodate said one or more instructions.

9. A computer program for performing correctness checks opportunistically, the computer program being embodied on a computer-readable medium, the computer program comprising:

a first code segment, the first code segment generating an initial instruction schedule from a first set of instructions, the first set of instructions including one or more instructions associated with a correctness check function;

a second code segment, the second code segment evaluating the initial instruction schedule to determine whether the initial instruction schedule includes spare instruction slots into which said one or more instructions associated with the correctness check function can be inserted; and

a third code segment, the third code segment inserting said one or more instructions associated with the correctness check function into the spare instruction slots if enough spare instruction slots exist in the initial instruction schedule to accommodate said one or more instructions.

10. The computer program of claim 9, wherein said one or more instructions associated with the correctness check function correspond to a conditional expression, and wherein prior to generating the initial instruction schedule, the first code segment performs initial code generation, wherein when initial code generation is performed, said one or more instructions associated with the correctness check function are separated from all other instructions of said first set of instructions so that the initial instruction schedule does not include any instructions associated with the correctness check function.

ABSTRACT

The present invention provides a method and an apparatus that enable spare instruction slots within a code module to be utilized opportunistically for insertion of instructions associated with correctness check functions. The apparatus of the present invention comprises a compiler, which may be comprised solely as hardware or as a combination of hardware and software. The compiler performs code generation and generates an initial instruction schedule. During the generation of the initial instruction schedule, the compiler ignores code sequences associated with correctness check functions. After the initial instruction schedule has been generated, the compiler examines the initial instruction schedule and determines locations of spare instruction slots in the initial instruction schedule that can potentially be utilized for insertion of the code sequences associated with the correctness checks. The code sequences associated with the correctness checks are then inserted into the instruction schedule to the extent that insertion of the code sequences does not lengthen the final instruction schedule. Consequently, no performance penalty is incurred at run time.

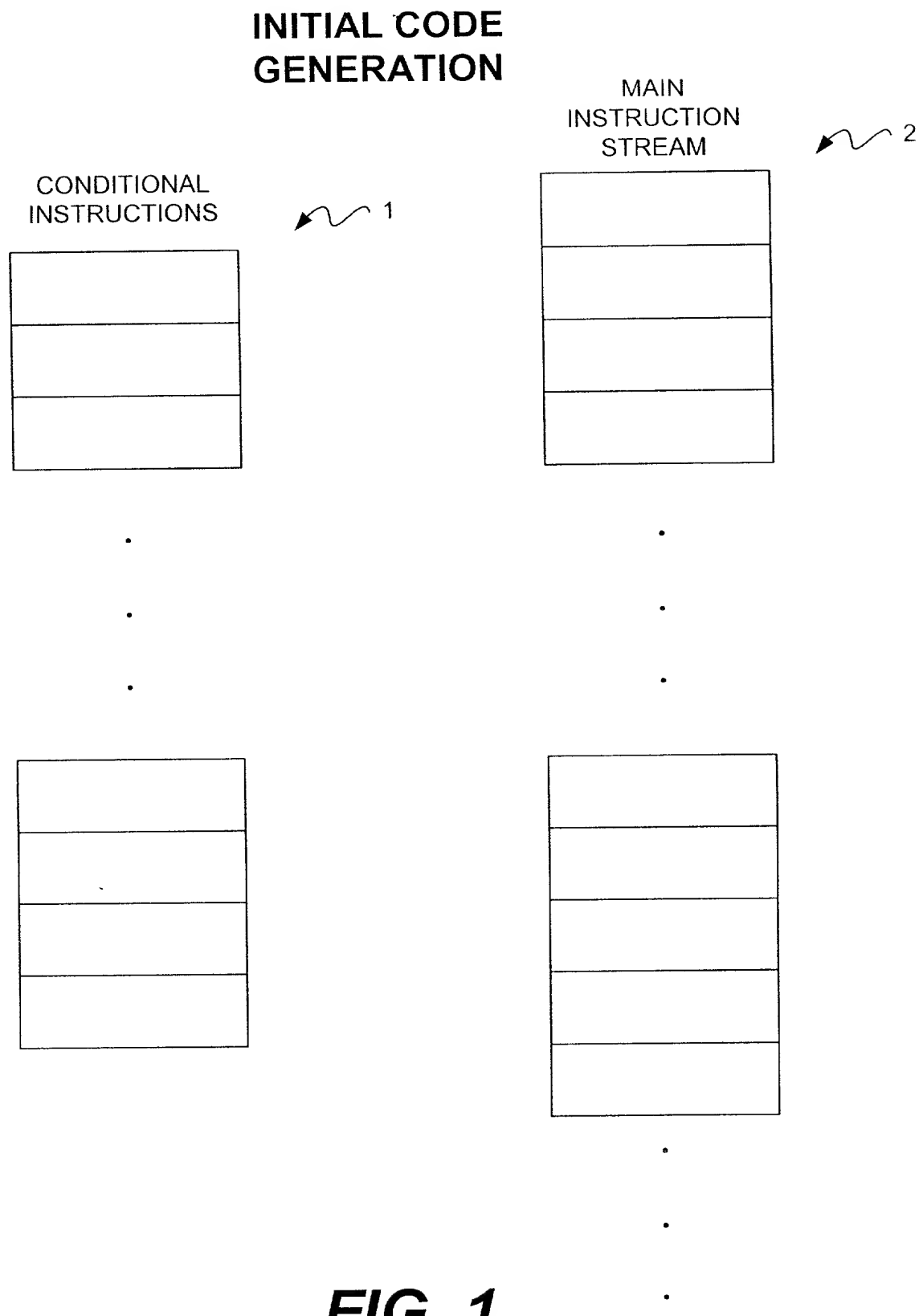
INITIAL CODE GENERATION
MAIN INSTRUCTION STREAM
2
CONDITIONAL INSTRUCTIONS
1

FIG. 1

INITIAL SCHEDULING

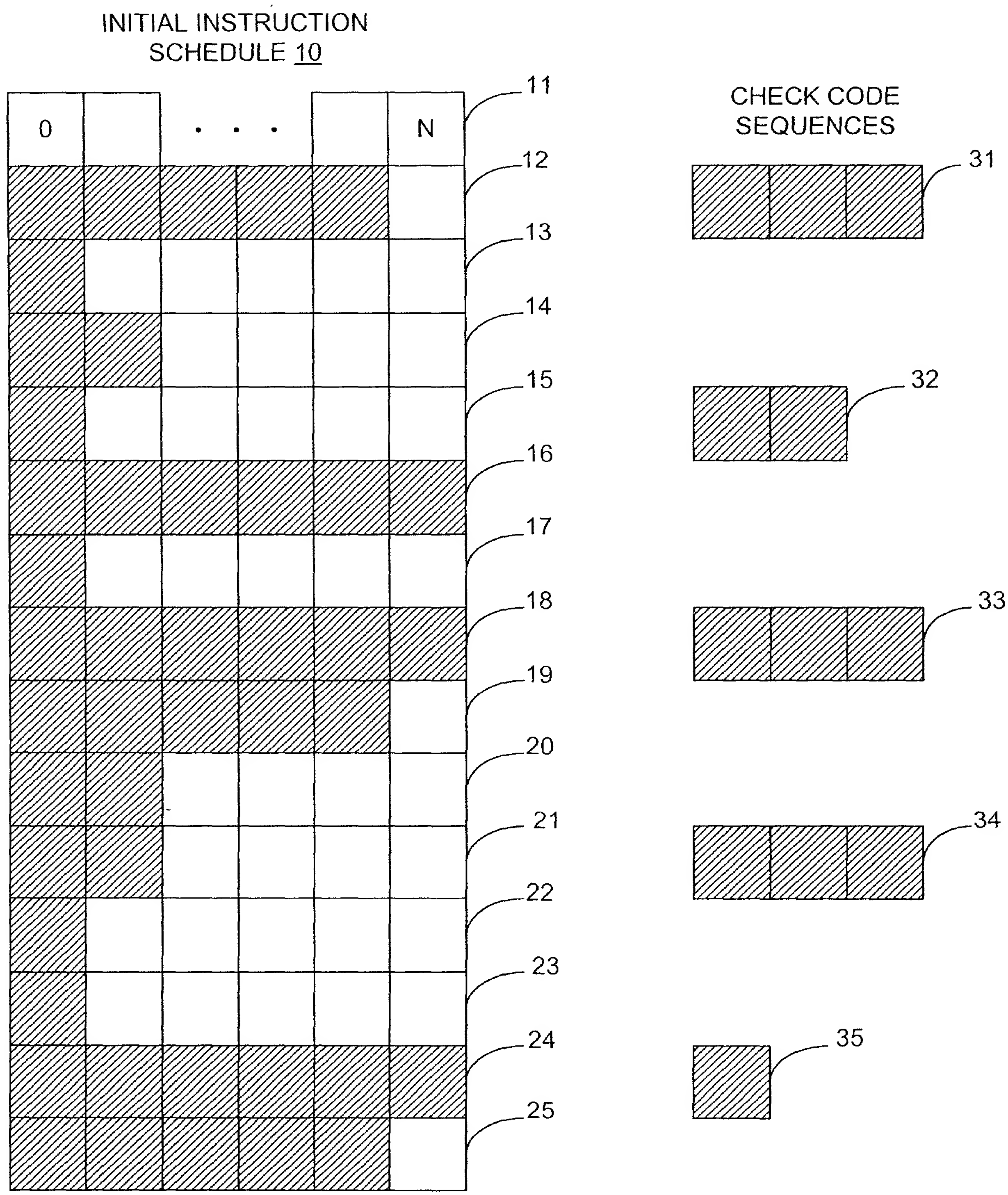

FIG. 2

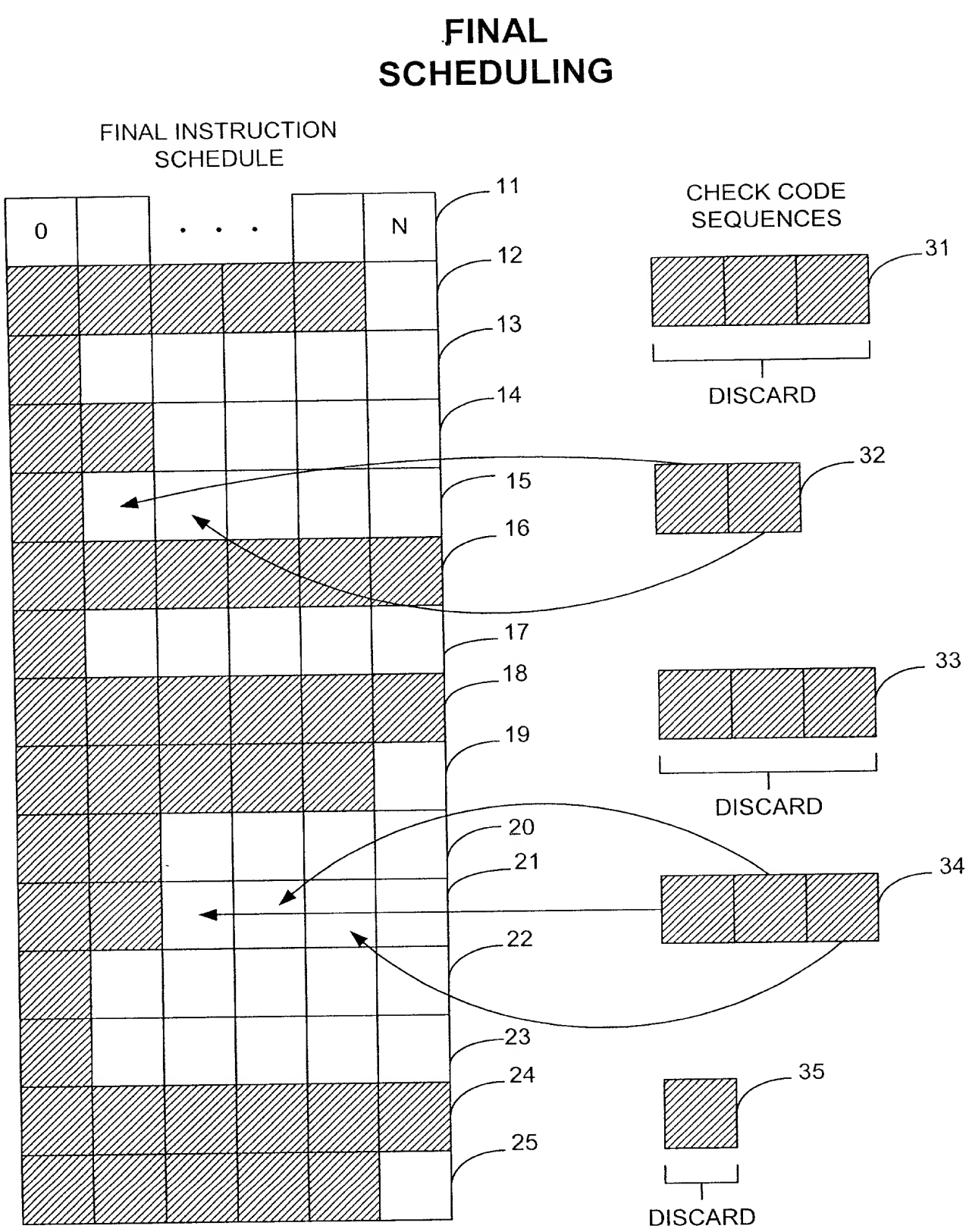
FINAL
SCHEDULING
FINAL INSTRUCTION
SCHEDULE
0
N
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
CHECK CODE
SEQUENCES
31
DISCARD
32
33
DISCARD
34
35
DISCARD

FIG. 3

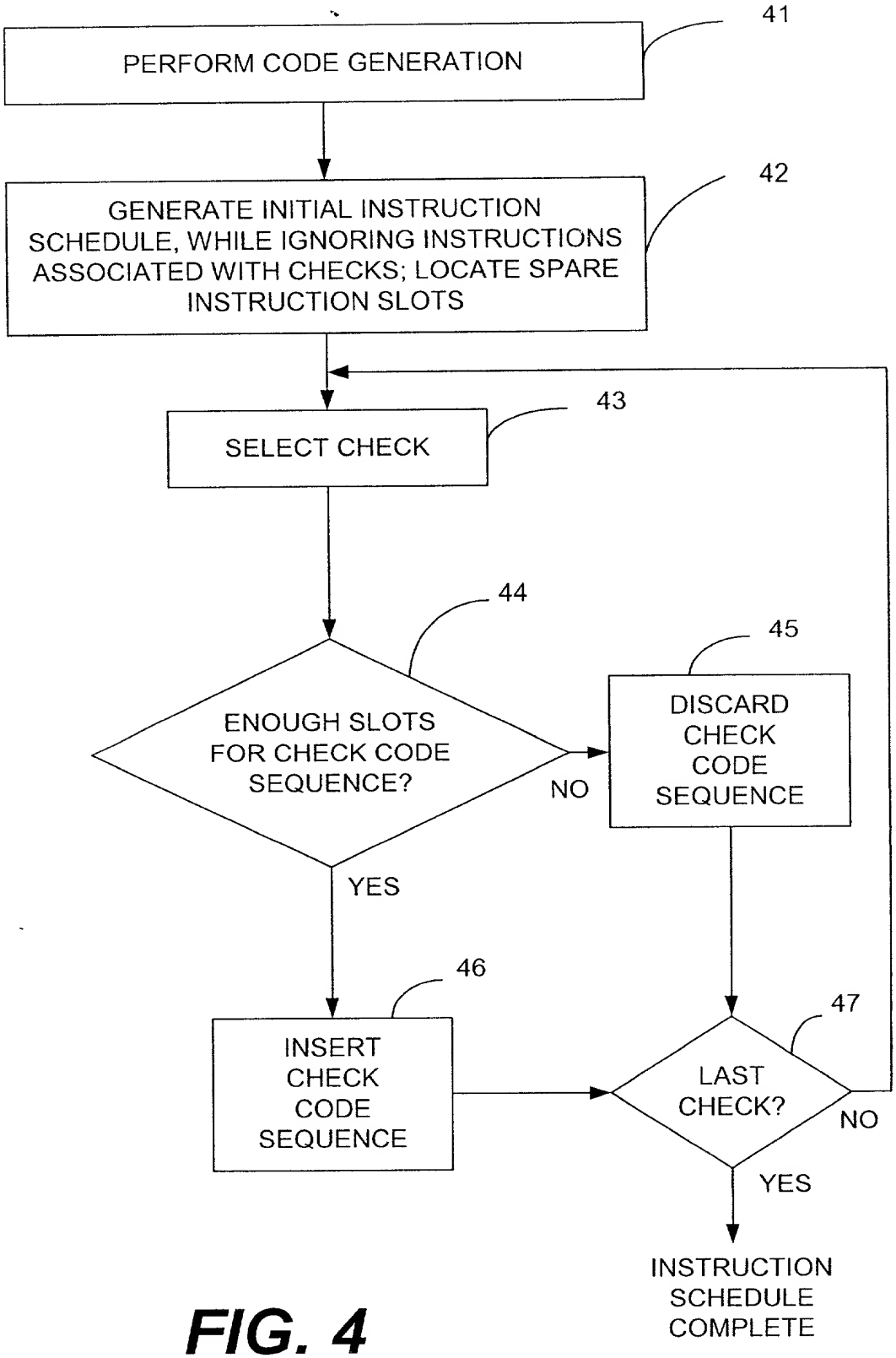
41
PERFORM CODE GENERATION
42
GENERATE INITIAL INSTRUCTION SCHEDULE, WHILE IGNORING INSTRUCTIONS ASSOCIATED WITH CHECKS; LOCATE SPARE INSTRUCTION SLOTS
43
SELECT CHECK
44
ENOUGH SLOTS FOR CHECK CODE SEQUENCE?
NO
45
DISCARD CHECK CODE SEQUENCE
YES
46
INSERT CHECK CODE SEQUENCE
47
LAST CHECK?
NO
YES
INSTRUCTION SCHEDULE COMPLETE

FIG. 4

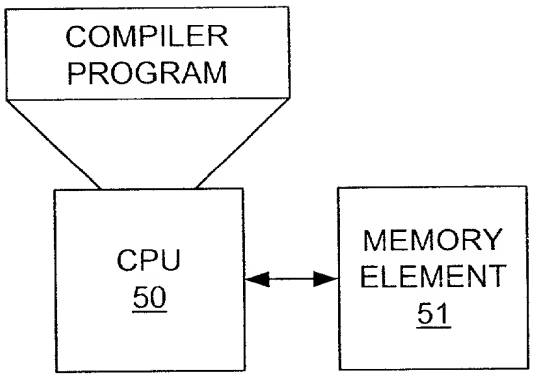

FIG. 5

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

ATTORNEY DOCKET NO. 10001151

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

Method and Apparatus for Performing Correctness Checks Opportunistically

the specification of which is attached hereto unless the following box is checked:

( ) was filed on _____ as US Application Serial No. or PCT International Application Number _____ and was amended on _____ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35 U.S.C. 119
			YES: ___ NO: ___
			YES: ___ NO: ___

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION SERIAL NUMBER	FILING DATE

U. S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION SERIAL NUMBER	FILING DATE	STATUS (patented/pending/abandoned)

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Customer Number 022879

Place Customer Number Bar Code Label here

Send Correspondence to:	Direct Telephone Calls To:
HEWLETT-PACKARD COMPANY Intellectual Property Administration P.O. Box 272400 Fort Collins, Colorado 80527-2400	Tuan V. Ngo (408) 447-8113

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Inventor: Carol L. Thompson **Citizenship:** US

Residence: 6937 Calabazas Creek Circle, San Jose, California 95129

Post Office Address: Same as Residence

Carol Thompson
Inventor's Signature

Nov. 13, 2000
Date